BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Car registration by document classification using computer vision

Supervisor
Lect. dr. Petrescu Manuela-Andreea

Author
Pop David Alexandru

2024

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Înmatricularea autovehiculelor prin clasificare de documente folosind viziune computerizată

Conducător științific
Lect. dr. Petrescu Manuela-Andreea

Absolvent
Pop David Alexandru

2024

Contents

Abstract

Nowadays, cars are almost necessary to live our lives, either to reach our daily jobs, either to travel on different places. But to be allowed to use your car, first you need to register your car. Since this is a extensive paper work, most of us forget some papers and we need to lose time to reschedule and try again to register the car. To avoid such situations, the solution is to develop a way to analyze documents at home, with a device with camera which most of use have, namely the phone.

The subject of this thesis is represented by the application in which users can scan their documents, get their types and ensure each folder to register a car is completed, everything from a user oriented mobile application, which supports different languages, Romanian and English and with in-app notifications to get the status of folder when creating a new one.

This thesis walk through each phase of development lifecycle, starting with research problem, presenting the existing methods with their advantages and disadvantages, moving on fixing the theoretical concepts to make grasping the presented solution easier and finally presenting the solution, architecture, technologies with their advantages and disadvantages and the results of classification, emphasizing each development stage. The solution chosen to be implemented in the system is one which does not require an extensive dataset of images, just a template and common text between all documents of that type and a confidence level associated with that text match.

The original contributions include a different OCR engine, more steps to compute the document's type to get better results, along with an implementation and design of the system for experiments. The developed application let user upload documents as images in the system and compute their type from a given list of templates using OCR, enhacement methods and regexes.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Chapter 1

Introduction

In an era where the volume of digital documents is expanding exponentially, the need for effective organization and classification methods is paramount.

The reason for conducting this thesis is document classification, the task of automatically assigning predefined categories to text documents, plays a pivotal role in numerous real-world applications such as registering a car.

The main problem and primary objective in this thesis is classifying documents using computer vision. Generally, there are two solutions: manual classification and automatic classification. The manual approach implies that the user specifies the type of document. The automatic approach makes use of the technology Optical Character Recognition, known as OCR. This approach does not come without problems. Text extraction is made using OCR engines. Such an engine is built to recognize characters in various fonts and sizes. Problems which might arise when using an OCR engine could be characters are not textual, hence using an OCR engine on an image is not enough to get the text. This system needs to consider different transformations to have better accuracy for character recognition. Such kind of transformations are: different qualities, blurring, warp perspective, rotating. The proposed solution warps the image and apply the above-mentioned transformations to get the text match and find the document from given templates. Another problem that arises is data privacy. The system uses all the privacy configuration needed from password hashing, session management up to image persistence on cloud services. All these features can be used using an android app interface written in Dart using the Flutter framework. The focus of this thesis is the system behind it which is exposed as an API written in object-oriented language, namely C#. Document classification microservice is written in Python using the OpenCV library.

This thesis is structured in five chapters, each one of them tackling a different step in the lifecycle of the system. In the second chapter we are going through the theoretical foundations of the solution implemented in the system. The context, explanations and usage of theses concepts are presented in details.

In the third chapter we are going through the already existing methods for document classification, including the solution implemented in the system. The motivation why this solution was implemented instead of the other is presented in this chapter. The explanation and trade-offs with advantages and disadvantages against other technologies and architecture decisions are presented in the fourth chapter.

In the fourth chapter we are also going through the implementation details, arguments for chosen technologies, testing and integration pipelines. Database normalization, diagrams and relation between tables is presented in this chapter.

We end this thesis with conclusions and future improvements for the presented system.

During the preparation of this work the author used Grammarly in order to fix grammar mistakes and ChatGPT to rephrase sentences with the purpose of improving the readability of the paper. After using this tools, the author reviewed and edited the content as needed and takes full responsibility for the content of the thesis.

Chapter 2

Theoretical foundations

In this chapter, we will go through the basics of computer vision and we will analyze the pattern matching way of solving this problem. More specifically we will focus on the getting biggest contour, similarities between images, warp perspective, enhancement methods and text extraction.

2.1 All contours

Contours are lines or curves that represents the boundaries of objects in a image. In order to determine the biggest contour, we must find all of them. The proposed solution make use of Canny edge detection function. The Canny edge detection algorithm, developed by John F. Canny in 1986, is a multi-step process for detecting edges in images. Key steps involve Gaussian blurring to reduce noise, gradient calculation to find edge strength and orientation, non-maximum suppression to thin edges, and edge tracking by hysteresis to link adjacent edges [LM18]. Before getting the contours, the image is converted to the gray color to simplify processing. We will use the *findContours* method from **OpenCV** [HBB+22]. The resulting image is then changed using Gaussian Blur. This blur helps to smooth out noise and reduce details. The threshold value for gradient magnitude are adjusted to fine-tune edge detection results using the trial and error method. The area of interest is the contour with the biggest area. This is done because is presumed that if user sent an image with a document then that document is object of interest in the image. Only the contours that contains four corners are considered possible candidates because most documents contains four corners [Figure].

2.2 Similarities between images

After getting biggest contour, the next step is to compare the images with template list to get the best match. Measuring similarity becomes simpler when two images differ only in specific pixels. One method to measure similarities between two images is using the Relative Average Spectral Score (RASE) [GASCG04]. The Relative Average Spectral Error helps us see how well different image blending methods work for each color band in the image. We are using RASE for the biggest contour to determine how similar it is with the template document, in the processing stage. For the current implementation, each document has an associated template, which was obtained from **Google Images**. For computing the RASE we will use the *sewar* library. The advantage of RASE is the computational efficiency in comparison with other metrics such as SSIM, Structural Similarity Index. It takes into account three aspects of the images: luminance, contrast, and structure. SSIM compares local patterns of pixel intensities that have been normalized for luminance and contrast. It is particularly effective at evaluating the perceived quality of images, which often involves considering how human visual systems perceive image differences [WBSS04].

2.3 Warping perspective

Sometimes the document represented by the biggest contour is not perfectly aligned with the template. In such situations we will warp the perspective from the uploaded image to the template's perspective. The image it's warped using perspective transformations matrices by using the *getPerspectiveTransform* and *warpPerspective* from **OpenCV**. The perspective transform is associated with the change in viewpoint. This process is very helpful when the document is not perfectly aligned with our templates, hence chances to get a good similarity score decrease. This operation is used after the region of interest(ROI) from that image is get, supposing the document is the ROI. This process is applied only after ROI is cropped from the image. It is worth mentioning that this process does not rotate the image. Below is the pseudocode of the algorithm described:

Algorithm 1 Pseudocode for Document Matching [IHHI22]

```
best_found_score ← 0
best_found ← null_object
for document_class in TEMPLATES_DOCUMENT_TYPES do
    best_align_score ← 0
    best_align ← null_object
    for threshold ← 0 to MAX_THRESHOLD_VALUE do
        biggest_contour ← get_max_contour(find_contours(uploaded_image, threshold))
        if biggest_contour ≤ 0 then
            continue
        end if
        transform_matrix ← cv2.getPerspectiveTransform(biggest_contour)
        image_warped ← cv2.warpPerspective(uploaded_image, transform_matrix)
        similarity_score ← get_match_score(document_page.template, image_warped)
        if similarity_score > best_align_score then
            best_align_score ← similarity_score
            best_align ← image_warped
        end if
    end for
    if best_align_score > best_found_score then
        best_found_score ← best_align_score
        best_found ← document_class.create_object(document_page, best_align)
    end if
    if best_found_score < MINIMUM_ALLOWED_SCORE then
        return NULL
    end if
end for
document_ocr ← empty map
```

2.4 Enhancement methods

Enhancement methods are useful to get better results for the OCR engine and for the RASE score. In this thesis, such methods are: grayscaling, Gaussian blur, dilation, erosion, HaarCascade frontal faces [PH19]. Grayscale images are simpler to deal with because they only show shades of gray instead of colors, making them easier to process, are faster to work with because there is less stuff to look at compared to color images and facilitate finding edges of contours. In the presented solution we will use the grayscale operation from the *OpenCV* library. Another method to enhance image is the usage of Haar Cascade for face detection in order to hide them in documents such as identity cards. A Haar-like feature considers neighbouring rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. An example of this would be the detection of human faces. Commonly, the areas around the eyes are darker than the areas on the cheeks. One example of a Haar-like feature for face detection is therefore a set of two neighbouring rectangular areas above the eye and cheek regions. The cascade classifier consists of a list of stages, where each stage consists of a list of weak learners. The system detects objects in question by moving a window over the image. Each stage of the classifier labels the specific region defined by the current location of the window as either positive or negative – positive meaning that an object was found or negative means that the specified object was not found in the image [Soo14]. Another enhancement method is blurring, namely using Gaussian blur. Blurring helps in reducing noise and detail in an image. It works by averaging the pixel values of an image with a Gaussian kernel, which is a matrix of values generated using a Gaussian distribution. In the proposed solution we will use the Gaussian blur method from *OpenCV*. Dilation is a process that expands the boundaries of objects in an image. It works by adding pixels to the boundaries of objects in the image based on a predefined structuring element. Erosion is the opposite process, shrinking the boundaries of objects in an image by removing pixels from the edges of objects. For both processes, we will use methods from *OpenCV*.

Chapter 3

Existing methods for document classification

In this section, we will explore two approaches of how we can classify documents. Firstly, we will review a machine learning method based on deep learning, then a computer vision method based on OCR and pattern matching using regular expressions.

3.1 Logistic regression

Logistic regression is one of the known and used supervised learning algorithms that are used to indicate the probability of a category is logistic regression. Given a document d represented by a set of features $x_1, x_2 \dots x_n$, the probability that $P(c_j|d)$ belongs to a class c_j is calculated as follows:

$$P(c_j|d) = \sigma(\sum_{i=1}^{n} x_i \times \omega_i) = \sigma(\omega^d x)$$

A text classifier is implemented in several steps. In the first step, labeled documents are represented as vectors of a certain length. The documents will be then filtered using data sanitization, such as removing unwanted characters, punctuation and symbols. Furthermore, the documents will be divided into two sets, the training set and the test set. The training set will used to fed into classifiers to train them, while the test set will be used to evaluate and predict the results [Sae22]. This method helps in text classification. That being said, the probability is obtained by linearly combining the set of features with a log functions. Thus, the $\sigma(x)$ or the sigmoid function is used in case of a binary classification problem.

$$\sigma(x) = \frac{1}{1+e^{-}x}$$

Because the target variables can only have two types, it is easy to model a relationship between the binary target variable and the predictor variables. This connection can be obtained using the sigmoid function.

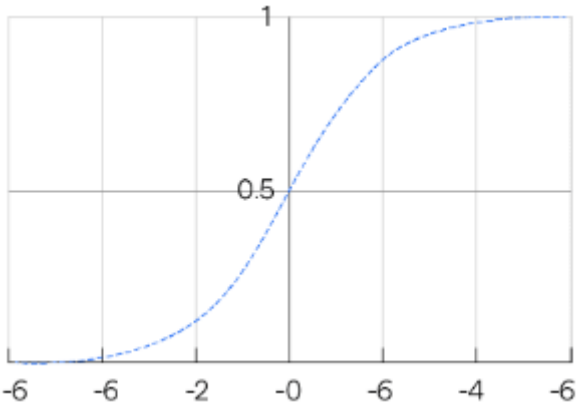

Figure 3.1: Sigmoid function

3.2 Bidirectional Encoder Representations from Transformers (BERT)

Until recently, the dominant paradigm in approaching natural language processing (NLP) tasks has been to concentrate on neural architecture design, using only task-specific data and word embeddings such as GloVe. The NLP community is, however, witnessing a dramatic paradigm shift toward the pre-trained deep language representation model, which achieves the state of the art in question answering, sentiment classification, and similarity modeling, to name a few. Bidirectional Encoder Representations from Transformers (BERT). It represents one of the latest developments in this line of work. It outperforms its predecessors, ELMo and GPT by a wide margin on multiple NLP tasks. This approach consists of two stages: first, BERT is pre-trained on vast amounts of text, with an unsupervised objective of masked language modeling and next-sentence prediction. Next, this pre-trained network is then fine-tuned on taskspecific, labeled data [ARTL19]. BERT, however, has not yet been fine-tuned for document classification. Why is this worth exploring? For one, modeling syntactic structure has been arguably less important for document classification than for typical BERT tasks such as natural language inference and paraphrasing. This claim is supported by that logistic regression and support vector machines are exceptionally strong document classification baselines. For another, documents often have multiple labels across many classes, which is again uncharacteristic of the tasks that BERT examines [ARTL19].

3.3 Computer vision and pattern matching

Modern OCRs make use of neural networks (e.g. PaddleOCR, Tesseract), these being trained, for example, to recognize entire lines instead of single characters. For our case, each document has a defined template which is loaded once when the server is started. The proposed solution for this thesis is to determine the set of all contours of the image. From the contours set, we extract the biggest rectangular area, representing the area of interest, and warp perspective of the contour with all our templates so that is perfectly aligned with the template. Following this, a serios of different image transformations are applied to the biggest contour such as blurring and grayscale. Given that certain documents, such as identity cards, may contain sensitive facial information, appropriate measures are taken to obscure these facial features utilizing the Haar-cascade algorithm [PH19]. To see how much looks like the current template we will use a quality metric, RASE (Relative Average Spectral Score) and save the best match, having the smallest RASE [IHHI22]. The motivation behind using this quality metric is, first of all, speed of computation and second of all, reliability, computing a good approximation if two images look the same. After all epochs, we will use an OCR engine to get the text from the best match with the current template and parse the text with a list of predefined regular expressions which have associated a confidence level. Finally, if the confidence level is greater than a threshold then we know the type of document.

Chapter 4

Design and implementation

In this section, the implementation will be explained from the backend up to frontend and infrastructure. Use case diagrams, UML diagrams and flow diagrams will be present, along with code examples, architecture decisions, programming languages, framework and trade-offs made during the implementation phase.

4.1 Used technologies

In this section, we will walk through the used technologies together with the programming paradigm, programming languages and frameworks used. The advantages of each one will be presented.

4.1.1 Object-Oriented Programming

Object-oriented programming, also known as OOP, is a programming paradigm that has the purpose of representing entities into objects and classes. OOP is a widely used paradigm [Bla13]. This paradigm focus on representing real entities into classes and objects, according to the 4 principles of OOP:

1. **Encapsulation** - is a principle that hides the implementation details of objects from the outside world. It states that all important data and functionalities are contained within the object; only selected data is available externally. Each object's inner workings and state are stored privately within the specified class, whereas other objects do not have access to them or the ability to make changes. Instead, they can only interact with a few public functions or methods. This form of data hiding provides program security and control over object state changes, reduces the risk of errors, and makes the program more understandable.

2. **Abstraction** - helps you focus on a entity's important elements and represents real world objects within the code. It allows you to construct more understandable programs. Abstraction can be thought of as an expansion of encapsulation.

3. **Inheritance** - is a principle in which new classes can be based on already existing classes. This principle offers a way to overriding, changing the implementation of the function, overloading with another arguments a function or method and altering parent class properties, if their visibility allows. This comes very handy when we are dealing with classes with similar purpose and prevents duplicate code and a ensure a common functionality among classes which extends the parent class. In this way we can reduce the cyclomatic complexity of the code and make it cleaner.

4. **Polymorphism** - is a principle which is strongly related to inheritance by allowing objects to behave differently by letting the developer to decide which method gets called. For instance, the ”drive” function can be used differently in different objects such as ”car”, ”bike” and ”motorbike”. In this way, we do not let the compiler decide which method to call, but developers decide. Furthermore, polymorphism makes the development process easier by allowing creation of common function to be used for multiple types of classes.

4.1.2 ASP .NET Core and C#

C# is a general-purpose, type-safe, object-oriented programming language. The target of the language is to increase productivity and to make a faster development process. To this end, C# balances simplicity, expressiveness, and performance. The chief architect of the language since its first version is Anders Hejlsberg (creator of Turbo Pascal and architect of Delphi). The C# language is platform neutral and works with a range of platform-specific runtimes [Alb21]. C# is a rich implementation of the object-orientation paradigm, which includes encapsulation, inheritance, and polymorphism. It comes with an Web API solution for backend, thanks to following advantages:

1. ASP .NET has many packages with maturity resulting in many out-of-box functionalities and variety to pick from for backend development.

2. It is very efficient for an API, being in the top most used programming languages for backend.

3. It is cross-platform, meaning it can run on Windows, Linux or macOS.

The following packages were used in the development of the backend solution:

1. **Entity Framework** - Is an ORM (Object-Relation Mapper) that lets you create tables from entities, queries using a fluent API for the data-access layer. The entities that are mapping to tables are called Data Access Objects, shortly DAO. The way to interact with database is using *DbContext* and to make changes to the database is by creating and applying new migrations. The approach used in this implementation was a *Code-First* approach, meaning that the tables were created by the entities from the codebase.

2. **Mapster** - Is an object-to-object mapper package. It makes very easy to transform from an object of certain type to another object of another type. It helps transforming from the request models to command or queries and then to the DAO type.

3. **Serilog** - Is a logging package. It makes very easy and handy to log the entire HTTP request. It does out-of-box the whole logging, providing multiple sinks. The one used in the solution is the console sink.

4. **Mediatr** - Is a package that implements the mediator design pattern. Together with the CQRS (Command and Query Responsibility Segregation) contitutes the basis for a so called vertical slice architecture. Every request that write to the database is a command, every request that get data from database is a query. Another benefit that this package offers is a called *Behaviours*. This behaviours is code executing before and after a certain command or query. In the solution proposed there are two behaviours used, namely the transaction behaviour which starts a transaction in the database for every command and commit if everything went successfully and another one, validation behaviour, which validates fields of requests. In case something does not obey the validation rules, an exception will be thrown resulting in a bad request with a status code of 400.

5. **SignalR** - Is a package that offers a wrapper around the WebSockets implementation in .NET. It makes the usage of WebSockets very approachable. A WebSocket is a protocol for real-time communication, providing a simultaneous two-way communication. It uses the Transmission Control Protocol (TCP) connection. The usage in this solution is to send notifications to the client about the status of classication process, since it might take some time to compute.

6. **NUnit and Moq** - Is a package for unit testing in .NET and mocking. Mocking is a way to change dependencies of a certain function since unit testing relies

on testing function in a isolated environment and considering all the dependencies returns the correct and expected values.

4.1.3 Python

Python is high-level, dynamically typed programming language. From the strongest characteristics of Python we can enumerate: easy to read and write, many packages with powerful functionalities and cross-platform. It was created by Guido van Rossum in the late 1980s. The following packages were used in the solution:

1. **OpenCV** - Is an open-source library for computer vision. This package was used for to solve the document classification problem. It contains a huge number of functions for image processing and enhancements methods.

2. **Flask** - Is a web framework which helps to build APIs. It supports HTTP methods and routing and JSON serializer. It is suitable for building small APIs and very easy to use.

3. **PaddleOCR** - Is an OCR engine to extract text from images. It also supports the Romanian language. It uses the the ppocr model as the default model. The engine is open-source, hence the classified documents are safe and there are no leaks.

4. **Sewar** - Is a package for image quality assessment using different metrics, RASE being the one used in the solution.

4.1.4 Flutter and Dart

Dart is a programming language designed by Lars Bak and Kasper Lund and developed by Google. It can be used to develop web and mobile apps as well as server and desktop applications [Nap19]. The advantages of using Dart:

1. **Easy to use:** - It is easy to use since its syntax is very similar to another programming languages such as C# or Java, Dart having a syntax based on C programming language.

2. **Performance:** - Dart compile the code to native platform such as Android or IOS for mobile development or to JavaScript for web-based applications.

3. **Asynchronous programming:** - Dart has support for asynchronous programming, making it very handy to handle operations such as network requests, user input, without blocking the main thread.

Flutter is Google's portable UI framework for building modern, native, and reactive applications for iOS and Android [Nap19]. Flutter uses Dart as programming language. The advantages of using Flutter:

1. **Cross-platform**: Flutter is suitable for building apps because it deploys on multiple platforms such as Android, IOS, web and desktop. This reduces the development time.

2. **Hot reload**: One the most interesting features is the Hot reload, meaning that developers can see the changes made in code faster in the emulator without compiling and building the project again. This accelerates the development time.

3. **Third parties**: Flutter offers a huge number of packages for different purposes. The ones used in this solution are: camera, provider, secure storage, dio, http, signalr, localizations.

The packages used in the Flutter solution:

1. **Camera:** - This package offers the API for accessing the camera on the device. This is give the opportunity to user to take pictures of the documents, but also give the possibility to upload images from its device image gallery.

2. **Secure storage:** - This package offers a safe and secure storage for saving sensitive data like the JWT after the user gets logged in. The way it's storing the data is of the form of key value pairs, similar to a dictionary, but the data gets encrypted using advanced cryptographic algorithms and it is also persistent, meaning that after the app gets closed, the data saved in local storage will persist after the application is launched again.

3. **Dio and http:** - This packages offers a way to send HTTP requests much easier compared with the standard API. Both are very useful for establishing communication with server.

4. **SignalR** - This package is a wrapper around the web socket implementation in Dart. It makes the integration with the WebSocket implementation on backend very easy.

5. **Localizations** - This package offers helper methods and classes with localization of user, meaning that user can change the language of the app. The supported languages in the project are: Romanian and English. The user can change the language during runtime and the app does not require to restart. The translation are found in files with the extension .arb (Application Resource

Bundle), which are very similar to .json files but they offer more flexibility and brings more feature such as placeholder, description, plural and gender. The translation files are generated from .arb files using the command line by typing *gen-l10n* in the root of the project.

4.1.5 PostgresSQL

PostgreSQL is an open-source relational database system with the purpose of persisting data on backend. It is widely used in development of applications. The following advantages leads to the decision to use PostgreSQL as database in the solution:

1. Most programming languages have integration with PostgreSQL.
2. It is open-source and is cross-platform, it can run on multiple operating systems such as Windows, Linux, macOS.
3. Offers a wide variety of features such as: complex queries, indexing, full-text search, bloom filter(is a space-efficient data structure that is used to test whether an element is a member of a set).

4.2 Backend

In this section we will walk through the backend part of the application, which includes the business logic namely user, folders and documents management together with interaction with document classification service.

4.2.1 Requirements elicitation

The main reason of requirements elicitation is to describe the set of features of the application. Every feature has a set of requirements that it must meet in order to be considered done and valid.

The next step is to describe the system in terms of use cases and actors. Accordingly to [BD09], actors are defined as the external entities which interacts with the system and the use case are sequences of actions between an actor and the system that describe a functionality. A use case can be split into several scenarios in order to describe multiple possible flows of the functionality.

The application main goal is to classify documents by images. That means user upload a series of images and the system determine the document type from each individual image using optical character recognition (OCR). Next we will define the functional requirements by describing the use cases and their scenarios in natural

language.

Upload image in the system

The user can upload multiple images. Images are stored in the cloud for storage. The classification is performed on given images and the result is returned with the found document types. Because this might be a long process, a notification system sends the status of request along with completed steps.

Authentication in the system

The user must be authenticated in the system in order to know it's identity. The authentication can be done in multiple ways, either using a classical login or register form by inserting the email and password, or an easier one using Google Sign-In.

Register

The user can register in the system using it's email and password. The email must respect the format of an email address and password should have at least five characters. The email must be unique and not be used by another user.

Login

The user can login the system using it's email and password. The account must exists before this action. If the account is taken by a sign-in with Google, an error will be thrown. The password criteria are similar with the register ones.

Sign in with Google

The user can login or register with Google account. In case the user it's already registered, it can not sign in with Google since email is already taken. This method makes an easy and fast way for user to logged into the app. If the account does not exists, it is automatically created after successfully login in for the first time.

Document management

The user can manage it's folders and documents by deleting already existing ones. This actions is irreversible and user must be aware of this aspect. An existing folder can not be updated after it is created, the only way is to create a new folder.

Delete a folder

The user can delete an already existing folder. This method requires the user to be logged in and it can only deletes only his personal folders. After successfully deleting a folder, this action can not be reverted and the folder is lost.

All the above mentioned are represented using the below use case diagram. There are two actors, one representing the human user and the other is the document classification service.

4.2.2 API structure

Application Programming Interface (API) is a set of rules for two or more computer programs to communicate with each other [Red11]. In the application, the API has

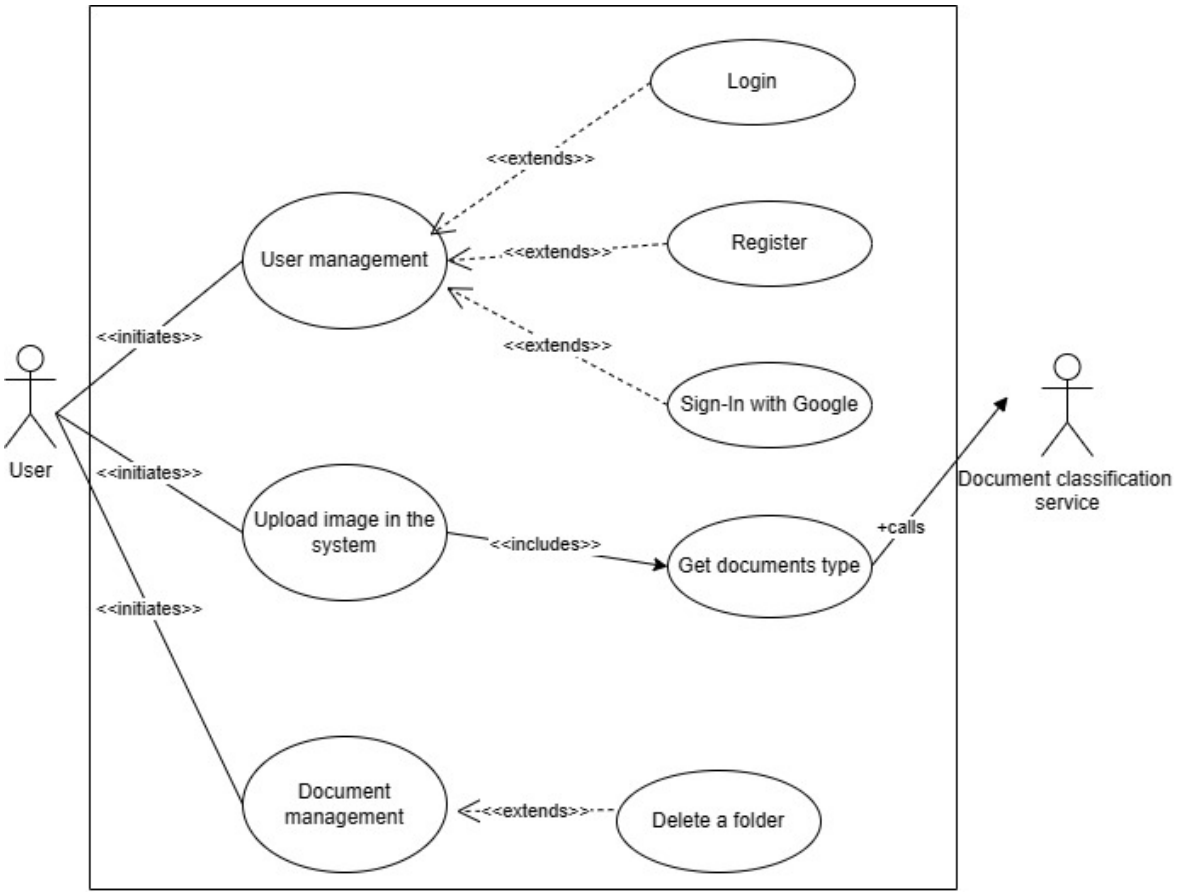

Figure 4.1: Use case diagram

multiple endpoints and respect the stardard of REST (Representation State Transfer). Only the authentication, login, register and sign-in with Google endpoints does not require the user to be authenticated. All of the endpoints stands between the route **/api**.

1. **/register** - This endpoint is used for introduction of a new user in the system. It takes an email and password as JSON in the body of the request. The method of this endpoint is POST. The password will be hashed using BCrypt. It returns the JWT if successfully registered with a status of 200, otherwise the error with the associated status code.

2. **/login** - This endpoint is used to check if an user exists in the system. It takes an email and password as JSON in the body of the request and returns the JWT if the user exists with a status code of 200, otherwise it returns 404. The verb for this endpoint is POST.

3. **/register/google** - This endpoint is used to authenticate an user using Google OAuth 2.0 protocol. It takes as parameter the token generated by OAuth after

successfully sign-in and register the user in the system if it does not exist. It returns a new JWT generated by the system. This verb for this endpoint in POST.


```
{
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJuYW1laWQiOiIyMWVmZWQ3OC04OWRjLTQ4NTEtYWJiMi1lMDBlMTNkNzhiMTIiLCJodHRw
        Oi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9zaWQi
        OiIyMWVmZWQ3OC04OWRjLTQ4NTEtYWJiMi1lMDBlMTNkNzhiMTIiLCJlbWFpbCI6InN0cmlu
        ZyIsIm5iZiI6MTcxMjI3MjkzNywiZXhwIjoxNzEyODc3NzM3LCJpYXQiOjE3MTIyNzI5Mzcs
        ImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6NTAwMC8iLCJhdWQiOiJodHRwczovL2xvY2FsaG9z
        dDo1MDAxLyJ9.TPsz1PBS1jrJPnnxKtOZEgD8dVzZwCt3H9-9RwJfzFk"
}
```


Figure 4.2: Response after *login*, *register* and *register/google*

4. **/profile** - This endpoint is used to get user data from the system. This endpoint get the user id from the JWT found in Authorization header of HTTP request. This endpoint requires the user to be authenticated. The verb for this endpoint is GET. If the user is not authenticated a 401 unauthenticated response is returned and if the user does not exists in the system it returns 404 not found.


```
{
    "email": "david@gmail.com",
    "role": 0,
    "authenticationType": 0
}
```


Figure 4.3: Response after */profile*

5. **/folder** - This endpoint using the verb POST creates a new folder in the system. The name, type and list of images is given in the body of the request using the Multipart Form Data. Images are saved in the cloud and passed to the document classification service. If the folder is created successfully, the status code 200 success is returned, otherwise the error code and the status code accordingly. The only allowed extensions for images are PNG, JPG, JPEG. This endpoint requires the user to be authenticated and the user id is get from JWT found in the authorization header of HTTP request.

```
{
    "id": "e322f3f3-554b-4c9f-8db6-2e72aa213b77",
    "name": "da",
    "type": 0,
    "isCorrect": false,
    "errors": [
        "9"
    ],
    "documents": [
        {
            "id": "0690fd88-8009-4076-ac3f-4a60f932bc47",
            "storageUrl": "https://documentsimages.blob.core.windows.net/images/
                e322f3f3-554b-4c9f-8db6-2e72aa213b77/
                0690fd88-8009-4076-ac3f-4a60f932bc47",
            "documentType": 0
        }
    ]
}
```


Figure 4.4: Response after */folder* with verb *POST*

6. **/folder/folderId** - This endpoint using the verb DELETE delete an existing folder with its associated documents from the system by getting the folder it from the endpoint route. It returns 200 success if the folder is deleted successfully otherwise it returns the error and associated status code. The user can only delete only its folders. In case the id is not passed in the route of the request or is not a valid GUID, a 405 method not allowed response is sent.

7. **/folder** - This endpoint using the verb GET gets all the folders of a user. The user id is taken from the JWT passed in authorization header from the HTTP request. If is successfully, it returns status code 200 with data, otherwise the error and associated status code. The output will be in the JSON format. Data returned in the body consists of the folderId, folder type, name, a boolean indicating if it is correct or valid, the associated errors if they exists, and the array of documents associated.

```
[
    {
        "id": "099bcad6-1df0-4c66-96e2-519f314e4177",
        "name": "asd",
        "type": 0,
        "isCorrect": false,
        "errors": [],
        "documents": [
            {
                "id": "658ce38f-d615-4ff9-a34b-06a3033f0177",
                "storageUrl": "https://documentsimages.blob.core.windows.net/images/
                    099bcad6-1df0-4c66-96e2-519f314e4177/658ce38f-d615-4ff9-a34b-06a3033f0177"
                "documentType": 0
            },
            {
                "id": "d13aefa0-5f7c-4c23-8a15-985baa90db50",
                "storageUrl": "https://documentsimages.blob.core.windows.net/images/
                    099bcad6-1df0-4c66-96e2-519f314e4177/d13aefa0-5f7c-4c23-8a15-985baa90db50"
                "documentType": 0
            }
        ]
    },
```


Figure 4.5: Result after */folder* with verb *GET*

4.2.3 System architecture

In this subchapter we will walk through the architecture of the system, especially the backend part.

The architecture choosen for the backend part is a modular monolith architecture. Monolith is a architectural style of building applications as a single unit, meaning the entire business logic and data access layer are tightly coupled and deployed as a single package. Modular monalith solve this problem by splitting the layers into modules. Each module have a single responsibility. The following modules were used to split the application: Domain, Application, Presentation, Infrastructure, Tests.

Module	**Explanation**
Domain	This is where are stored common entities and classes across the application. We will find here: data access objects (DAO), custom exceptions classes such as ValidationException, NotFoundException, DuplicateEntryException, constants such as error messages, enumerations and interfaces such as ISoftDelete for soft deleting fields and IAudit for audit fields.
Application	The layer which handles the business logic. Since the architecture uses *Mediatr*, the commands and queries and their associated handlers are found in this layer. Another very important gather from this layer is DTOs (Data Transfer Objects). The role of DTOs is to send data from application layer to the presentation layer. The *Mediatr* package offers a feature called *Behaviours*, which is code that can be execute before and after a command or query. The behaviours present in the solution are *TransactionalBehaviour* which makes every command to execute every database operation in a transaction. If everything executes successfully, behaviour will commit the transaction, otherwise it will rollback.
Presentation	This layer is responsible for handling the requests, responses along with authentication and authorization. Before a request gets into the system, it passes through a series of middlewares. In the solution there is only one custom middleware, ExceptionMiddleware, which is responsible for handling exception throwned by the system and returning a standard response to the client. Here we will also find the configurations such as dependency injections, database, Swagger, authentication & authorization, websockets and services. Controllers are responsible for handling incoming requests and response. In the system we will find three controllers, namely Folder Controller, User Controller and Document Controller. The endpoints found in these controller were presented above, along with their responses.
Infrastructure	This layer is responsible database context and migrations. A migration is created when we want to change database schema. Migrations are created using the CLI (Command Line Interface) of the Entity Framework. Here we will also find a concept called *Interceptors* which will intercept database queries. In the system are two incerceptors, *AuditInterceptor* which will intercept when a new entity which inherits from *IAudit* interface is created and will populate those fields. The second interceptor is *SoftDeleteInterceptor* which will change the fields of entities which inherits from *ISoftDelete* interface and fill those fields.

Tests	This layer contains the tests which ensure the correctness of the system. There are two types of tests here: unit tests and integration tests. Unit tests are useful to prove the correctness of a part of a system in a isolated environment. On the other hand, integration tests check if components works correctly combined.

Table 4.1: Modules in application

The architecture of the system discussed above can be seen in Figure 4.6.

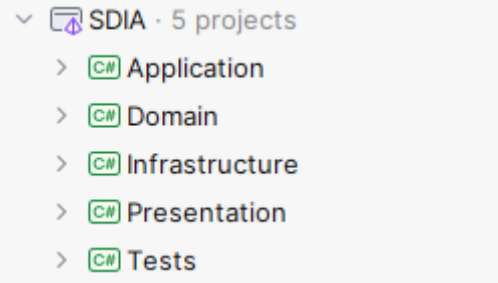

Figure 4.6: Modular monolith architecture

This vertical slice architecture have different advantages, such as: better responsibility segregation between layers and better scalability because one layer have minimal impact on others. This is a so-called *Clean Architecture* [Mar17]. This separation can be seen in Figure 4.7, where we have domain in center, being covered by application layer and lastly by presentation and infrastructure, which are on the same layer.

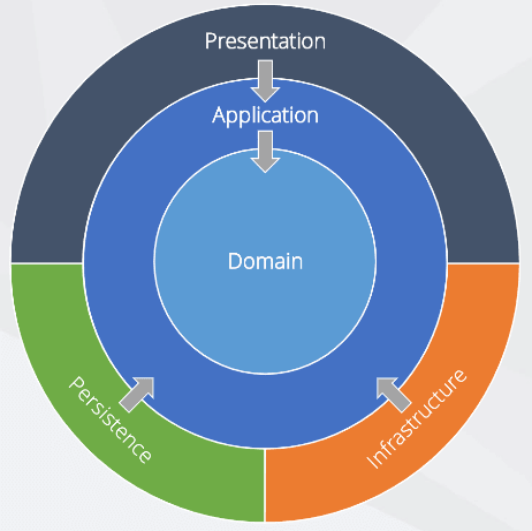

Figure 4.7: Clean architecture layers

The advantages of this type of architecture are: components are testability, business logic can be tested independently of external resources, maintainability, changes are less likely to propagate unintendedly in the code and scalability, by allowing components to be modified, added, or disrupted without modifying the existing structure [Mar17].

The system is connected to a relational database, PostgreSQL, and it's diagram is represented in Figure 4.8. The database contains four tables: Users, Folders, Documents and FolderErrors. One user can have many folders, while a folder can have multiple documents and multiple errors. The database is in the second normal form (2NF). For a database to be in 2NF, it must first be in 1NF (first normal form). To be in the first normal form, every attribute must be atomic. To be in the second normal form, it must be first in 1NF and every attribute which is not a primary key must be fully functionally dependent on the primary key of the relation.

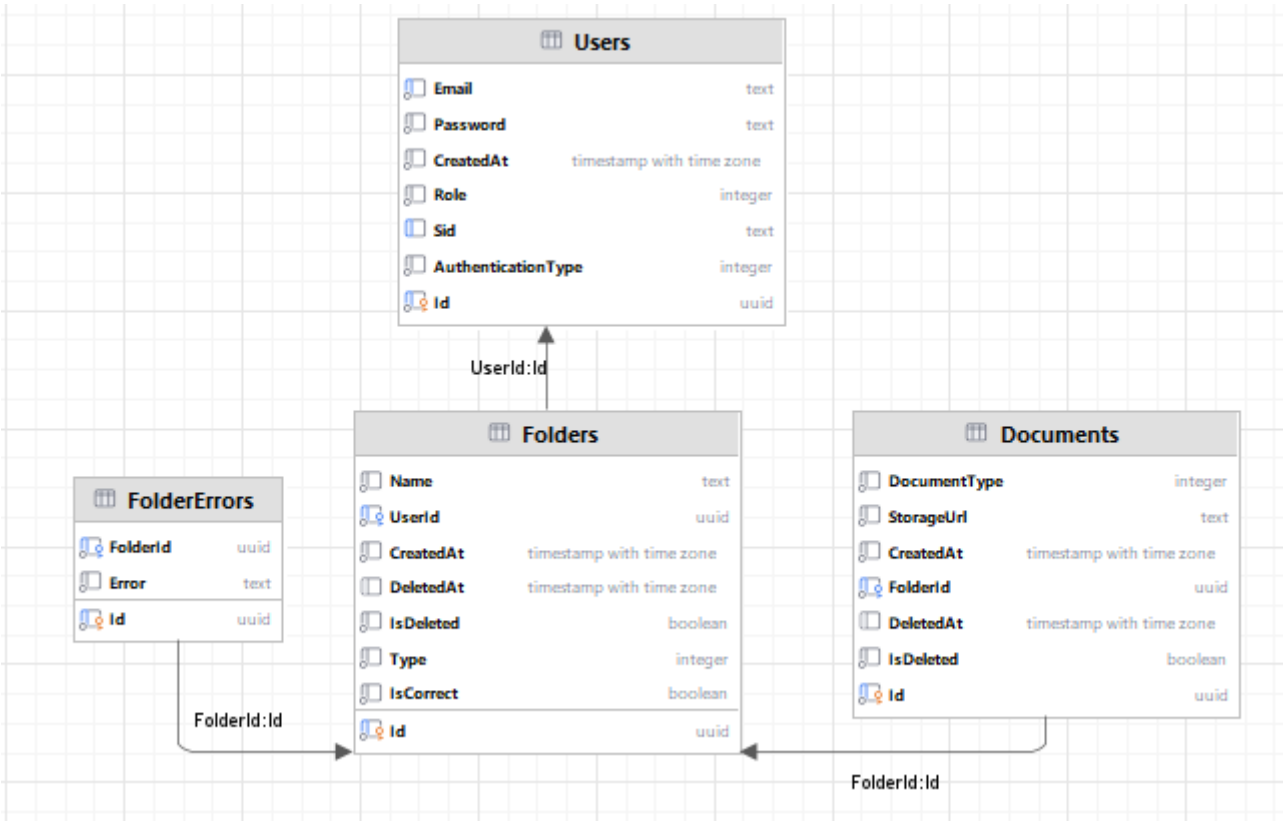

Figure 4.8: Database architecture diagram

4.2.4 Implementation

In this subchapter we will go through the code flow of each endpoint, up from the controller, through the database and back. We will start with User Controller. Below is depicted the class diagram of User Controller and associated classes:

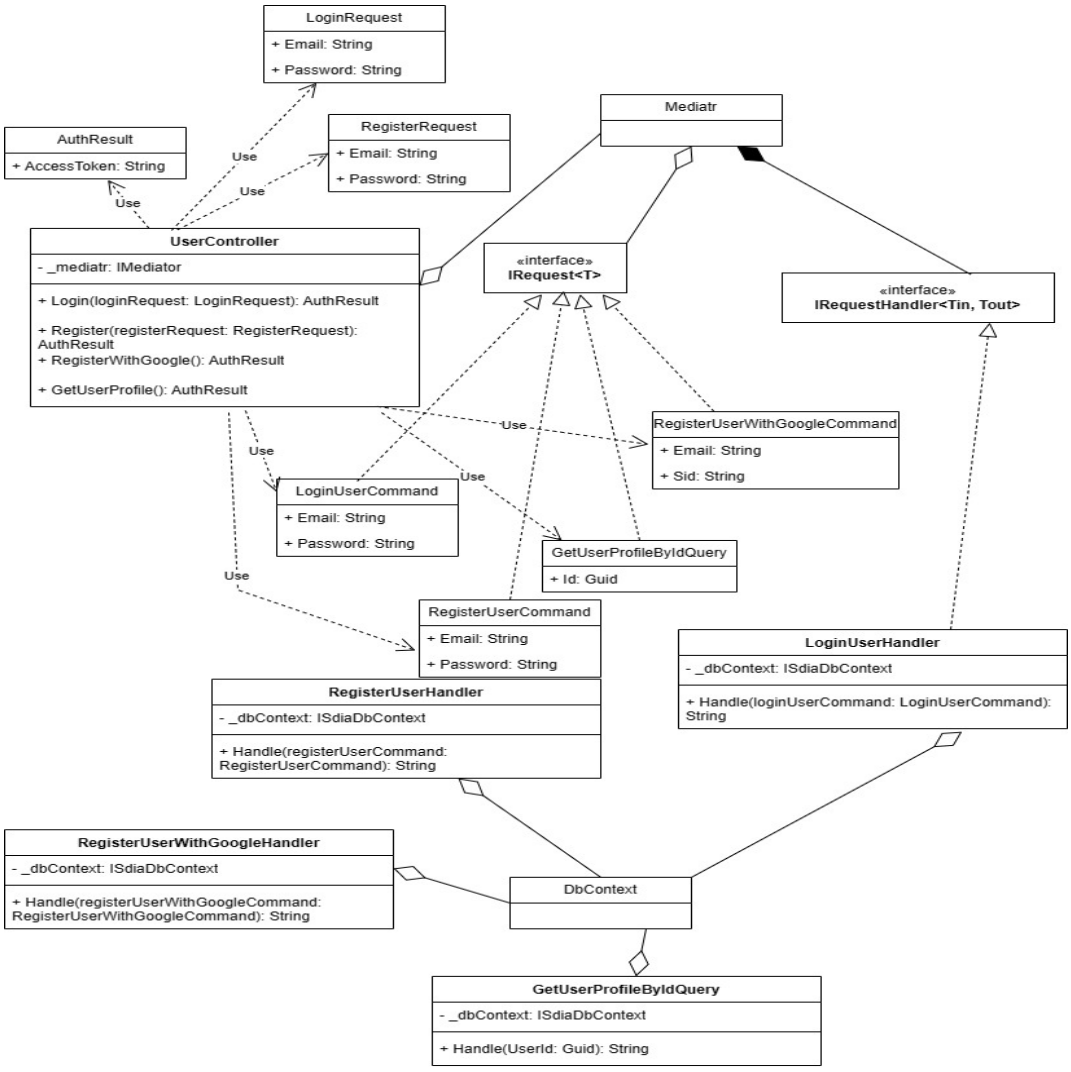

Figure 4.9: Class diagram of User Controller

In the *Login* and *Register* endpoints, an UserCredential model is read from the body of the request, then it's created a new LoginUserCommand or RegisterUserCommand which next is sent to the Mediatr. In both handlers, validations are made according, such as user does not exists or exists, password matches. All these operations are made to the database with the help of *ISdiaDbContext*, which gets injected when the class is created by Dependency Injection. If the authentication is successfully made, a JWT token is created containing, besides other fields, the user id and email. If some validation errors occurs, a specific exception, such as: DuplicateEntryException in case of already existing user in Register with an already existing user, or NotFoundException in case if the user does not exists while logging in, are thrown and caught in error middleware. The JWT is returned to the controller, which returns a new AuthResult object containing the token as response with a status code of 200.

In the *GetUserProfile* endpoint, authorized, the user id is taken from the JWT

passed in request header. Then a query is created with the user id and send to the Mediatr. In the handler is checked if the user id exists in the database, if it does not exists a NotFoundException is thrown, otherwise the user is retrived from database and an UserDTO is returned from the handler. Database context is injected through the interface ISdiaDbContext in the constructor. After this, controller returns a status code of 200 with the UserProfile as response. The mapping between UserDTO to UserProfile or from User to UserDTO is made easier with the package *Mapster*.

In the FolderController, to create a new folder we get the CreateFolderRequest from form. This entity is composed of name, folder type and a list of documents, every document being is composed of an image. Then a CreateFolderCommand is created and passed to the Mediatr. Inside the CreateFolderHandler, four dependencies are injected, namely: ISdiaDbContext which is responsible for interaction with database, IImageService which is responseible for saving images into cloud using a package from Azure Blob, IDocumentService which is responsible for sending HTTP requests to the document classification service in Python and ICreateFolderNotification which is responsible for sending status of the process to the client with the help of Websockets. All the operations, namely upload images in cloud and document classification are done in parallel, to get a better time performance. At the end a FolderDTO is returned to the controller. Next, controller will create a new command, AnalyzeFolderDocumentsCommand, to analyze errors in a given folders such as missing documents. In the handler, documents in a folder are analyzed according to known folder type and known documents for each folder type. At the end, an AnalyzeFolderDto is send to the controller with the errors, in case they exists. Controller will make a last query to get the folder with errors from database and return the response to the client with a status code of 200.

Because there are many ways in which a image is analyzed, there are much chances to generate an already computed contour. To avoid such situations and also to decrease the computational time, a caching mechanism has been used. The mechanism is a dictionary from stardard library in *Python*. After the biggest contour is computed, is saved in the cache. The code which exemplifies this functionality is found in Figure 4.10.

```
biggest, max_area = self.biggest_contour(contours)
if biggest is not None:
    biggest = self.reorder(biggest)
    tuple_biggest = tuple(tuple(item) for item in biggest.reshape(4, 2))
    if tuple_biggest in cache:
        continue
    cache[tuple_biggest] = True
```

Figure 4.10: Biggest contour caching

For example, in Figure 4.12, we can see how the image is warped according to the template and the identity card, which is the region of interest, is extracted from the Figure 4.11, which is the input. The output image is in the grayscale format to help the OCR Engine to better extract the text from the image.

Figure 4.11: Input identity card

Figure 4.12: Output identity card

```
await Parallel.ForEachAsync(request.Documents, cancellationToken,
    body: async (d :CreateDocumentDto, forEachCancellationToken) =>
{
    var name = $"{folder.Id}/{d.Id!.Value}";
    var uri :string = await _imageService.UploadImageAsync(name, d.File,
        forEachCancellationToken); // Task<string>
    Interlocked.Increment(ref uploadedDocuments);
    if (uploadedDocuments == request.Documents.Count)
    {
        creatingFolderNotification.ImagesUploaded = true;
        await _createFolderNotification.SendNewStatus(creatingFolderNotification,
            folder.UserId,
            cancellationToken); // Task
    }
```

Figure 4.13: Create folder handler code example

As we can see in the Figure 4.13, each document is using a different thread to be saved in cloud and analyzed. Since this is a parallel execution for multiple documents, an interlocked increment has been used to increment the number of uploaded documents to send the notification to the user, in order to avoid the racing condition. This interlocked operation works similarly with a mutex(mutual exclusion). The notification is send with the new status of operations at the user with id who creates the folder.

In the DeleteFolder endpoint, the folder is is passed as route parameter. After this, a command to delete folder is created and passed to the Mediatr. In the handler, is checked if the folder exists and it belongs to the user who is making the request. If not, a NotFoundException or NotAuthorizedException are thrown. Otherwise, folder is deleted and controller returns a status code of 200 to the client. Database context gets injected through ISdiaDbContext interface in the constructor of the handler.

In the GetAllFolders endpoint, user id is taken from the JWT passed in the header of request. User can only query its own folders. A query to get folders of an user is created in the controller and passed and all folders of that user are returned as a list of FolderDto. Database context gets injected through ISdiaDbContext interface in the constructor of the handler. Then the controller will adapt that list to a FolderInfoResponse list and return the list together with a status code of 200 to the client.

4.2.5 Testing

The testing phase of the application is an important phase in the lifecycle of the product to ensure the correct functionality of the components and the obey the acceptance criteria.

The approach used in the solution is with *unit-testing*. We define a scenario and expected results and test the command against that scenario. We will introduce a testing technique called *mocking*, which is meant to replace any external dependencies of the component under test with objects on which we can define the behaviour.

The scenario being tested in the application is the *CreateFolderHandler* with an empty image and all types of folders. All the external services, such as *ImageService*, *DocumentService* and *CreateFolderNotification* are mocked.

Another type of testing is using a manual approach, namely manual testing. In the manual testing phase, the system is tested agains exploratory testing, which means the application is tested without predefined test cases, to identify potential issues. These tests have been applied both on frontend and backend of the system.

Figure 4.14: Integration tests

As we can see in Figure 4.14, for the validation of the system, integration tests have been used to test how components interacts with each other. The purpose of integration tests is to check that components communicate between them. In context of integration testing, there are two main definitions namely: **Stub** - is a piece of code that simulates the behavior of a lower-level module (called by the module under test). **Driver** - is a piece of code that simulates the behavior of a higher-level module that calls the module under test. In context of the provided solution, the driver

is the *FolderController*, while the stubs are the *GetFoldersByUserIdQueryHandler* and *ISdiaDbContext*.

4.3 Mobile application

As mentioned previously, Flutter is used as frontend framework. For each request made by the frontend, excepting the authenticating requests, the JWT will be passed in the Authorization header of the request. The frontend is diveded into multiple screens: *Login*, *Register*, *Home*, *Create folder* and *Settings*. For a better re-usability, some components have been extracted to individual widgets: *Are you sure modal*, *Folder card*, *Navigation bar* and *Login with Google button*.

In order to state using the app, we must create an account or sign-in using Google. After successfully login we will be redirected on the main page. On the bottom of the screen we find the navigation bar together with the screen we are looking at hovered. On the home page we can find already existing folders and their type, together with the delete folder option. After clicking on a a folder we will go at a new screen in which we will see the folder details along with documents in that folder and validation errors, in case they exists. If we want to delete an existing folder, a pop-up will open to confirm our intention. This operation is irreversible, hence once deleted, it can not be recovered. The login page and home page are represented below, in Figure 4.16 and Figure 4.15:

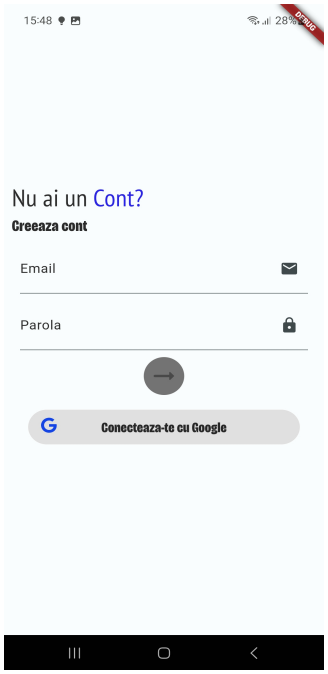

Figure 4.15: Login page

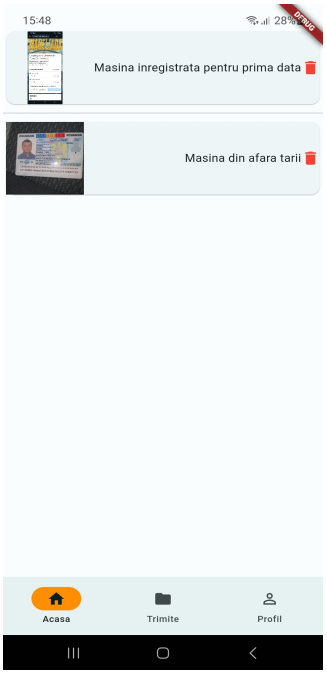

Figure 4.16: Home page

To create a new folder we press on the middle button in the navigation bar and

a new screen will appear. Here we can choose from the dropdown the type of the folder. From the *Add image* button we can choose between taking a picture with camera and upload one from phone's gallery. In case we are do not like one image, we can delete it by pressing the delete button on the right of the image. After pressing *Send* button on bottom on the page, a pop-up will open indicating the status of our request. There are two operations which happens, one which save the documents in the cloud for storage and future retrieval and the second one which does the document classification from our images. Because this is might be a long process, a *Websocket* is used to indicate the status of the above mentioned operations. After the folder is created successfully, the user is redirected on the main page and can refresh the list of folders to see new added folder. In case of any errors, such as not found documents, they will be printed when the user clicks on the folder. These screens are represented below, in Figure 4.17 and Figure 4.18:

Figure 4.17: Folder screen with complete one

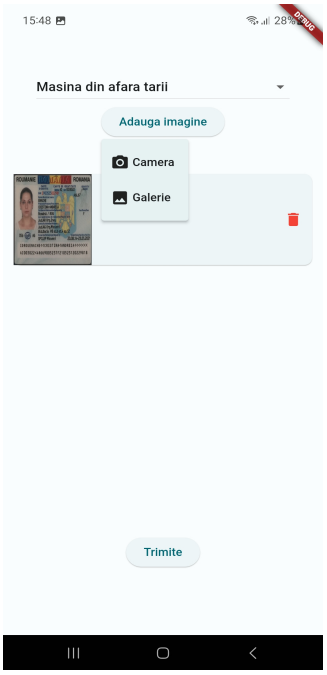

Figure 4.18: Create folder screen

In case we want to sign out from our account, or to change the language into English, we can do these on the *Settings* screen, the last one in the navigation bar. The languages supported by the application are: *Romanian* and *English*. There is no need to restart the application to change the language. The screen is depicted in Figure 4.19.

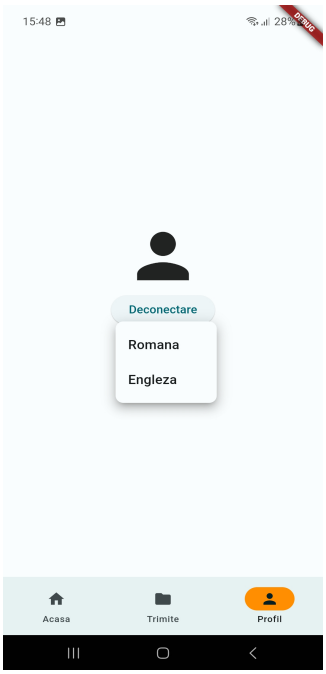

Figure 4.19: Settings screen

Change language feature is done with the help of the package *Localizations*, presented above. In order to determine the text for our locale, we have two .arb files responsible to handle translations. Instead of writing hard coded strings as text in code, we will use **Locale** class and variables generated from our .arb files. The Romanian and English translations files can be found in Figure 4.20 and Figure 4.21.

```
"helloWorld": "Buna ziua!",
"forgotPassword": "Ai uitat parola?",
"uploadPhoto": "Incarca imaginea",
"retakePhoto": "Anuleaza incarcarea",
"submit": "Trimite",
"cancel": "Anuleaza",
```

Figure 4.20: .arb file for Romanian language

```
"helloWorld": "Hello World!",
"forgotPassword": "Forgot Password?",
"uploadPhoto": "Upload Photo",
"retakePhoto": "Retake Photo",
"submit": "Send",
"cancel": "Cancel",
```

Figure 4.21: .arb file for English language

4.3.1 Models

Because data must travel from frontend to backend and vice-versa, some documentation about entities is necessary. The Data Transfer Objects that are used in both frontend and backend are detailed below:

1. AuthResult - It is responsible to return the token after a successfully login or register.
2. CreateFolderNotification - It is the entity responsible for notifications sent by the Websocket when creating a new folder with the completed operations, store images in cloud and classify documents.
3. Document - It is the class responsible for storing all the data for an existing document such as image URL, document type and Id.
4. ErrorDetails - It is responsible for handling any kind of errors from the backend, including different validations errors. The backend will always return the same error object, thanks to the error handling middleware.
5. Folder - This class contains data for an existing folder, including its documents. User can retrieve only its folders.
6. UserProfile - This class contains user information, such as email and role. Right now, role does not have an usage.

4.4 Infrastructure

In this subchapter we will talk about infrastructure of the project, namely contanerization of environment and pipelines for continous integration, their advantages and problem which solves.

4.4.1 Docker and Docker Compose

Docker is a platform that allows you to ”build, ship and run any app, anywhere”. It has come a long way in an incredibly short time and now is considered a standard way of solving one of the costliest aspects of software development: deployment.

Before docker came along, the development pipeline typically involved combinations of various technologies for managing the movement of software, such as virtual machines, configuration management tools, package management systems and complex webs of library dependencies [MS19].

In the solution, Docker was used to containerize both backend services, the one in .NET and the one in Python. Dockerfile has been written for both services. For

the service written in .NET, all the *.csproj* files have been moved inside container to install all the dependencies. After this, all builds output, using the command *dotnet build*, are moved inside the container, which will produce four *.dll* files. In order to start execution of the backend inside the container, we use the command *dotnet Presentation.dll*, because the Presentation layer has no other dependencies and is the one which contains the startup configurations files. For the Python sevice, used for document classification, an image with Python 3.9 is used, and then all the packages from the requirements.txt are installed. In order to get the OCR engine working, an update to system is required and *python3-opencv* is required to be installed. Finally, the flask server is started and ready to accept incoming requests.

In order to avoid communication problems between containers and to start two separated containers individually, this is where docker-compose comes in, because it solves this problems. To allow services to communicate with each other, are placed inside the same network. Another container which is used, is the database container with *PostgreSQL*. Another concept which is used, is the *volume*, which is a way of storing data after the container is closed, otherwise everything stored in database will be lost. An exampled is showed in the Figure 4.22.

```
1  version: "3.8"
2
3  networks:
4    dev:
5
6  services:
7    api:
8      image: docker.io/library/backend
9      container_name: SDIA_backend
10     depends_on:
11       - "db"
12     ports:
13       - "5176:80"
14     build:
15       context: ./SDIA/
16       dockerfile: Dockerfile
17
18     environment:
19       - ConnectionStrings__SDIA=User ID =postgres;Password=postgres;Se
20       - ASPNETCORE_URLS=http://+:80
21       - DocumentServiceConfiguration__baseUrl=http://document_analyze
22     networks:
23       - dev
```

Figure 4.22: Docker compose file with backend and network

All the services are described under the *services* tag. *Depends-on* is a tag for a container to wait until another container is created and to do not be created before it. *Ports* is a tag for defining inside and outside ports. In the above example, 5176 is the port for communication from outside with the container, while 80 is the binded port inside the container for the api service. The network in which containers are created is called *dev* and is meant to allow communication between containers. In order to configure different environment variables before starting the containers, we use the *environment* tag. In the above example, the connection string to the database, the URL and the port for starting the application and the URL for document classification service are passed.

4.4.2 Continuous Integration

In this subchapter we will walk through the concept of Continuous integration, the problem which solves and how it is integrated in the solution.

Continuous practices, i.e., continuous integration, delivery, and deployment, are the software development industry practices that enable organizations to frequently and reliably release new features and products [SABZ17]. In the solution, one pipeline have been used, which builds the application and run all the tests. In this way, every time when a new feature is done and a pull request is opened, this steps, if failed, will block merging the pull request until issues are fixed. In the below presented exampled is a continuous integration pipeline from the solution using *Github Actions*. The code used in the system is depicted in the Figure 4.23.

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
    env:
      DOTNET_INSTALL_DIR: 'dotnet'
    strategy:
      matrix:
        dotnet-version: [7.0.x]
    defaults:
      run:
        working-directory: ./backend/SDIA

    steps:
      - name: Copy repository
        uses: actions/checkout@v4

      - name: Install dotnet
        uses: actions/setup-dotnet@v3
        with:
          dotnet-version: ${{ matrix.dotnet-version  }}

      - name: Check installed/cached dotnet version
        run: dotnet --version

      - name: Check build
        run: dotnet build

      - name: Check tests
        run: dotnet test
```

Figure 4.23: Continous integration pipeline

The steps for the above pipeline are: selecting the branches on which we want to apply the pipeline, the actions on which we want to apply the pipeline (push, pull request) and then defining the jobs. First, we will define the operating system on which we will run the application, then clone the repository, install dotnet with desired version, in this case 7.0 and at final we will run build and test commands for our programming language using the CLI. In this case, *dotnet build* and *dotnet test*.

Chapter 5

Conclusions and future work

Following the completion of the paper, the overall objectives have been achieved. In this work, it is presented a solution to classify documents when there are none datasests available with documents.

In the proposed solution, several concepts are applied from computer vision, up to software engineering. The application enhance user manangament, document manangement, security, cloud and document classification. The integration between systems, backend and frontend went without any problem and the application is easy to use for customers. Each screen was developed to create a flawless user experience and they do what they were meant to do in an acceptable amount of time. User experience was an important aspect during development time, that's why features such as loading screens, notifications and change languages in Romanian or English have been added.

Application can be improved to increase the overall user experience. The system provided is good enought, is not perfect and can be improved at different services.

The template mechanism is not a generic one, everytime a new document is added in the system, it must be added by a programmer in the code. The template must be a generic one, to get a good match with uploaded documents. That being said, the way of computing the similarities between two images can be changed using different approaches and apply different enhancements methods to image to speed up the process and get better results for OCR.

OCR engine is doing a great text extraction, but it works slower compared with other engines. Future work should focus on getting text much faster with the same accuracy, in this way the execution time of requests will decrease and will offer users a better experience. Such kind of engines are changed frequently, or others appears and future versions of the system should consider updating the engine.

The number of pattern matching using regexes and their confidence level associated should be increased to match many more possibilities. This feature is strongly related to the OCR engine, because the better with reality the text is, the more confi-

dence we have a document is of a certain type.

The security of the app should not be negligible and obey General Data Protection Regulation (GDPR) laws, considering the fact we are working with personal documents. Such improvements might be: using HTTPS(HyperText Transfer Protocol Secured) as communication protocol and a safe cloud provider for storage.

To conclude, the system is capable of recognizing documents from images with a given list of templates and patterns. Finally, according to uploaded documents, we can determine if the user can register a can and if he has all the necessary documents.

Bibliography

[Alb21] Joseph Albahari. *C# 9.0 in a Nutshell*. " O'Reilly Media, Inc.", 2021.

[ARTL19] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398*, 2019.

[BD09] Bernd Bruegge and Allen Dutoit. *Bernd Bruegge, Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition Publisher: Prentice Hall, Upper Saddle River, NJ, 2009; ISBN: 0-13-606125-7*. 10 2009.

[Bla13] Andrew P. Black. Object-oriented programming: Some history, and challenges for the next fifty years. *Information and Computation*, 231:3–20, 2013. Fundamentals of Computation Theory.

[GASCG04] M. Gonzalez-Audicana, J.L. Saleta, R.G. Catalan, and R. Garcia. Fusion of multispectral and panchromatic images using improved ihs and pca mergers based on wavelet decomposition. *IEEE Transactions on Geoscience and Remote Sensing*, 42(6):1291–1299, 2004.

[HBB+22] Mohammed Khalid Hossen, Sayed Bari, Partho Barman, Rana Roy, and Pranajit Das. Application of python-opencv to detect contour of shapes and colour of a real image. pages 20–25, 05 2022.

[IHHI22] Cosmin Irimia, Florin Harbuzariu, Ionut Hazi, and Adrian Iftene. Official document identification and data extraction using templates and ocr. *Procedia Computer Science*, 207:1571–1580, 2022. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 26th International Conference KES2022.

[LM18] Ruiyuan Liu and Jian Mao. Research on improved canny edge detection algorithm. *MATEC Web of Conferences*, 232:03053, 01 2018.

[Mar17] Robert C Martin. Clean architecture, 2017.

[MS19] Ian Miell and Aidan Sayers. *Docker in practice*. Simon and Schuster, 2019.

[Nap19] Marco L Napoli. *Beginning flutter: a hands on guide to app development*. John Wiley & Sons, 2019.

[PH19] Adri Priadana and Muhammad Habibi. Face detection using haar cascades to filter selfie face image on instagram. In *2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT)*, pages 6–9, 2019.

[Red11] M. Reddy. Api design for c++. *API Design for C++*, 01 2011.

[SABZ17] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017.

[Sae22] Nawar Khalid Saeed. Machine learning for document classification of financial documents with focus on invoices. 2022.

[Soo14] Sander Soo. Object detection using haar-cascade classifier. *Institute of Computer Science, University of Tartu*, 2(3):1–12, 2014.

[WBSS04] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

List of Figures